Microsoft
.net developer
training

Configuring and Deploying .NET Applications

Agenda: Day 2

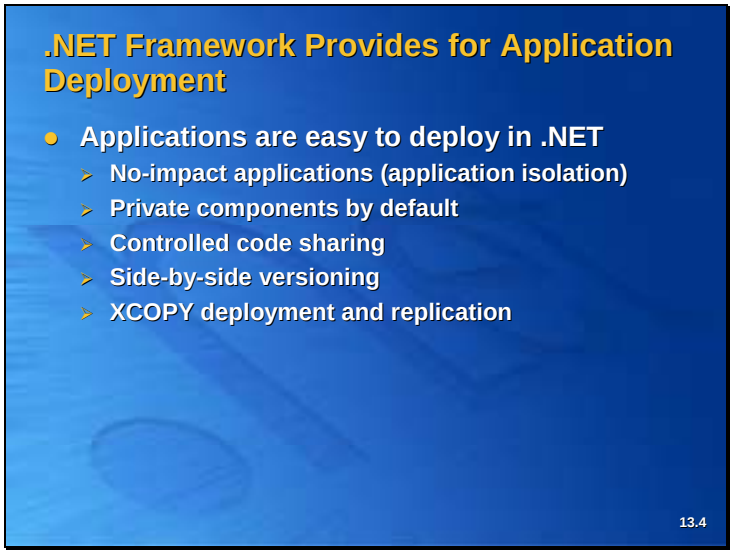
- Module 09: Creating XML Web Services
- Module 10: Code Instrumentation and Middle Tier Technology
- Module 11: Security in .NET
- Module 12: Lifecycle Tools for .NET
- ▶ Module 13: Configuring and Deploying .NET Applications
- Module 14: Application Interoperability
- Module 15: Migrating Applications to .NET
- Module 16: Building Applications for Mobile Devices

 ©2002 Microsoft Corporation

Module 13 Agenda

What you will learn:

- How .NET provides for deployment and things to keep in mind as you build your assemblies
- How applications are configured in .NET (both Windows and Web Applications)
- Deployment options for .NET Applications



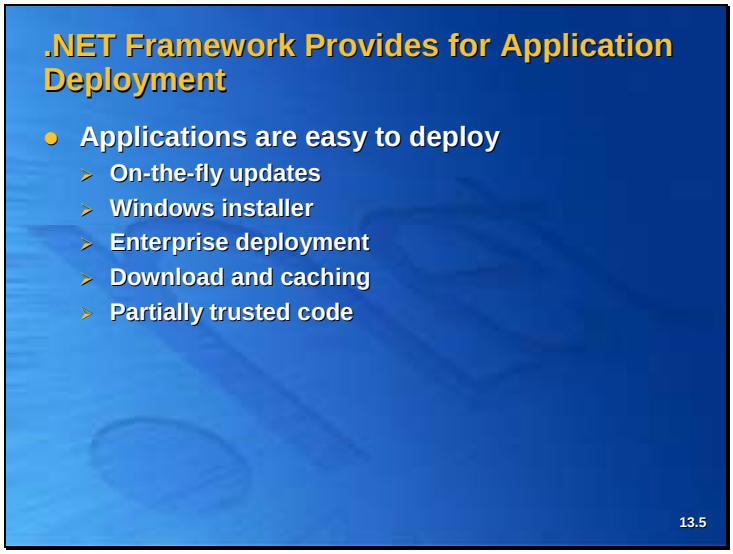
.NET Framework Provides for Application Deployment

The .NET Framework provides a number of basic features that make it easier to deploy a variety of applications. These features include:

- **No-impact applications** - Provide **application isolation** and eliminate DLL conflicts. Components do not affect other applications
- **Private components by default** - Components are deployed to the application directory by default and are only visible to the containing application
- **Controlled code sharing** - Sharing is explicit rather than default
- **Side-by-side versioning** - Multiple versions of a component or application can co-exist, you can choose which versions to use and versioning policy is enforced by the Common Language Runtime
- **XCOPY deployment and replication** - Self-described and self-contained components and applications can be deployed without registry entries or dependencies.

More details on deployment are covered later in the module.

 ©2002 Microsoft Corporation



.NET Framework Provides for Application Deployment

- **On-the-fly updates:** Administrators can use hosts such as ASP.NET to update program DLLs, even on remote computers.
- **Windows Installer:** Integration with Windows Installer makes advertisement, publishing, repair and install-on-demand available when you deploy your application.
- **Enterprise deployment:** Provides easy software distribution, including Active Directory
- **Download and caching:** Incremental downloads keep downloads smaller and components can be isolated just for use by the application for zero-impact deployment.
- **Partially trusted code:** Promotes identity based on the code rather than the user, policy set by the administrator and no certificate dialog boxes.

See the Appendix on .NET Security Advanced Topics for more on Code-based Security

Packaging Applications

By using the **.NET Framework**, developers can package their applications with the following options:

- As a **single assembly** or as a collection means you can use the .DLL or .EXE files as they were built
- As Cabinet **(CAB) files** means you can compress files into .cab files to quicken distribution or download
- As a **Windows installer package** or in other installer formats means you can create .msi files for use with the Windows installer or you package your application for use with another installer

 ©2002 Microsoft Corporation

Assemblies

Assemblies
Assemblies are the building blocks of the .NET Framework applications that form the fundamental unit of deployment, version control, reuse, activation scoping and Security permissions. Assemblies require no registry entries.

Deployment unit
When an application starts, only the assemblies the application initially calls must be present. Other assemblies, such as localization resources or assemblies containing utility classes, can be retrieved on demand. This allows applications to be simple and thin when first downloaded.

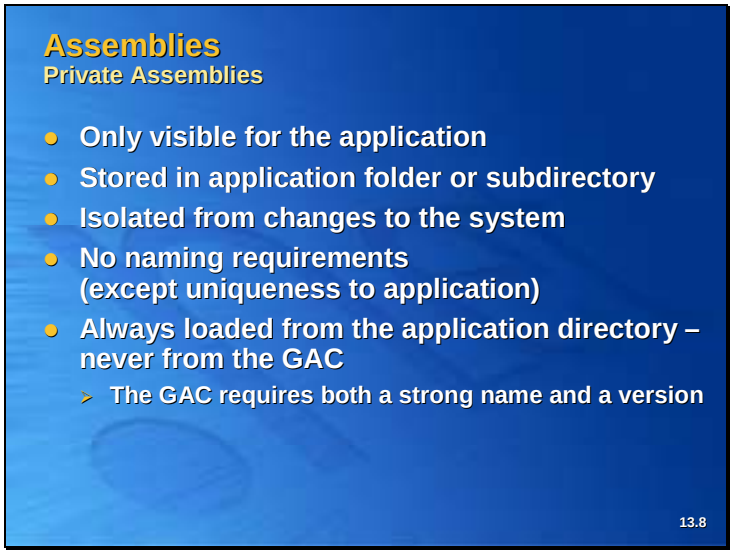
Security boundary
An assembly is the unit at which permissions are requested and granted. See module 11A: .NET Security Advanced Topics for more on Code-based Security.

Type boundary
Every type's identity includes the name of the assembly in which it resides.

Version boundary
The assembly is the smallest versionable unit in the Common Language Runtime; all types and resources in the same assembly are versioned as a unit. The assembly's manifest describes the version dependencies you specify for any dependent assemblies.

Side-by-side execution
The unit at which side-by-side execution is supported**.**

More information on side-by-side execution can be seen at:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguidnf/html/cpconsidebysideexecution.asp

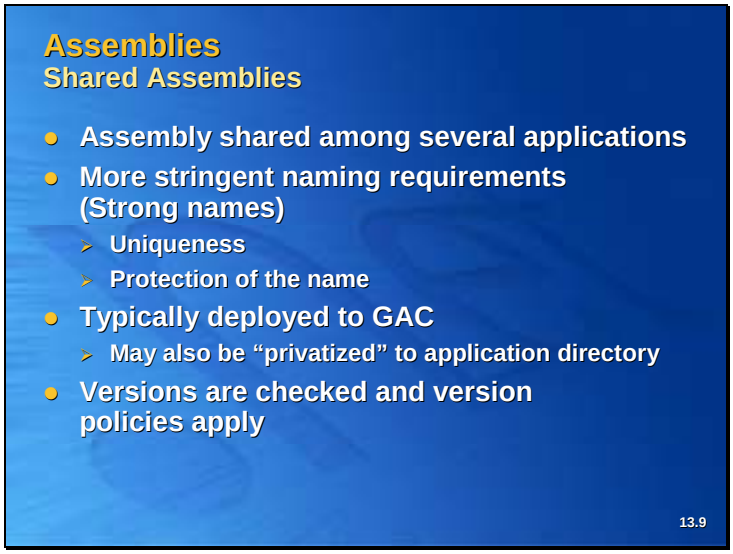

Assemblies
Private Assemblies
Only visible for the application
Stored in application folder or subdirectory
Isolated from changes to the system
No naming requirements
(except uniqueness to application)
Always loaded from the application directory –
never from the GAC
The GAC requires both a strong name and a version
13.8

Assemblies

Private Assemblies

To install a private assembly, both the application EXE and the assembly containing imported COM types must be installed in the same directory structure.

 ©2002 Microsoft Corporation



Assemblies

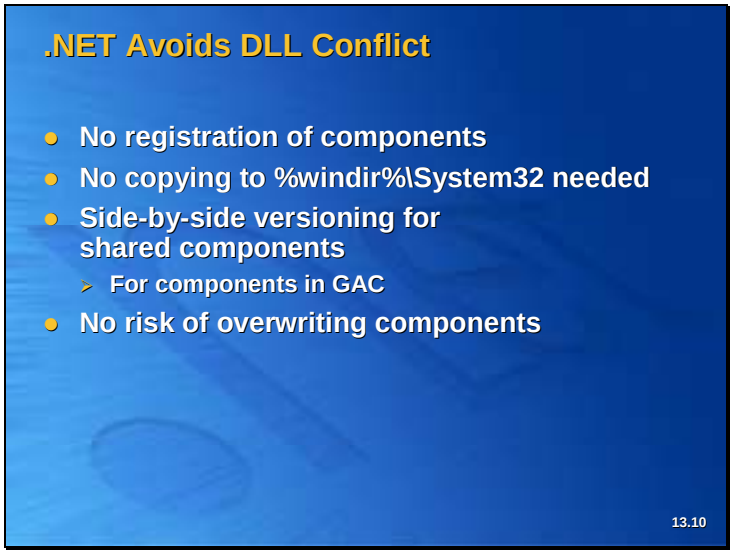
Shared Assemblies

A **strong-named assembly** includes the publisher's public key, which provides a unique identity.

Assemblies shared by **multiple applications** should be installed in a centralized repository called the **Global Assembly Cache** (GAC). However, you are not required to use the GAC for shared assemblies.

.NET clients can access the same copy of the assembly, which is signed and installed in the GAC. It is preferable to obtain the primary assembly from the publisher of the type library when one is available.

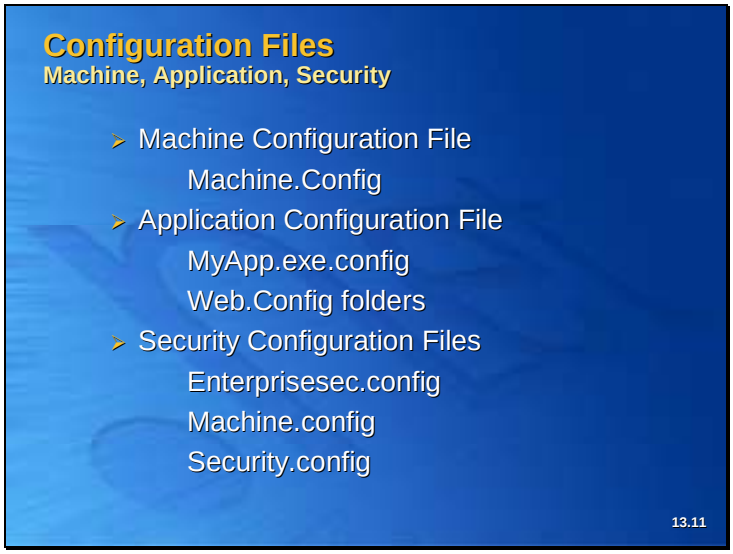
Version numbers are recorded in the assemblies and checked by the CLR at runtime, which allows the CLR to enforce versioning of assemblies. The configuration files also can be used to specify which version of an assembly will work best.


.NET Avoids DLL Conflict
No registration of components
No copying to %windir%\System32 needed
Side-by-side versioning for shared components
For components in GAC
No risk of overwriting components
13.10

.NET Avoids DLL Conflict

The CLR uses assemblies to continue toward a complete solution to DLL conflicts.

 ©2002 Microsoft Corporation



Configuration Files

Machine, Application, Web

Configuration files are changeable and parsable XML files that are case sensitive. They allow developers to change settings without having to recompile applications as well as allowing administrators to set policies that affect how applications run on their machines.

Managed code can use the System.Config APIs to read settings from the configuration files, but not to write settings to the files.

The **ASP.NET** configuration system features an extensible infrastructure that enables you to define configuration settings when your ASP.NET applications are first deployed. This allows you to add or revise configuration settings at any time with minimal impact on operational Web Applications and servers. It enables configuration information to be stored in XML-based text files so that any standard text editor or XML parser can create and edit ASP.NET configuration files.

Multiple configuration files, all named Web.Config, can appear in multiple directories on an ASP.NET Web Application server. This topic is discussed in more detail on slide 15.

At run time, ASP.NET uses the configuration information provided by the Web.Config files in a hierarchical directory structure to compute a collection of configuration settings for each URL resource. The resulting settings are then cached for all subsequent requests to a resource.

ASP.NET detects changes to configuration files and automatically applies new configuration settings to Web resources affected by the changes. The server does not have to be rebooted for the changes to take effect. Hierarchical configuration settings are automatically recalculated and re-cached whenever a configuration file in the hierarchy is changed.

The ASP.NET configuration system is extensible. You can define new configuration parameters and write configuration section handlers to process them.

ASP.NET protects configuration files from outside access by configuring Internet Information Services (IIS) to prevent direct browser access to configuration files. HTTP access error 403 (forbidden) is returned to any browser attempting to directly request a configuration file.

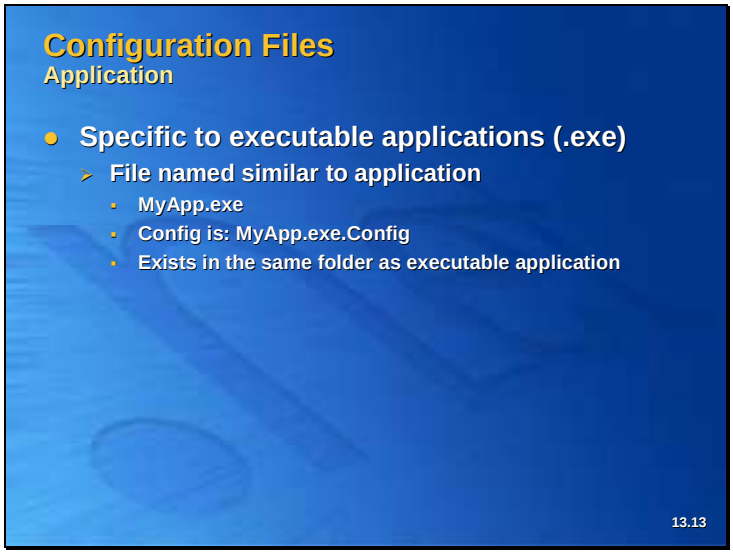
 ©2002 Microsoft Corporation

Configuration Files

Machine.Config

The configuration system first looks in the **machine configuration** file for API and ASP.NET settings. It then looks in the application configuration file. To keep the machine configuration file manageable, it is best to put settings in the application configuration file. However, putting the settings in the machine configuration file can make your system more maintainable. For example, if you have a third-party component that both your client and server application uses, it is easier to put the settings for that component in one place.

The machine configuration file includes settings that apply for the entire machine. The file is located at %runtime install path%\Config\Machine.config. Machine.config contains information such as machine-wide assembly binding.



Configuration Files

Application

Application configuration files contain settings specific to an application. These settings include assembly binding policy, remoting objects, custom channels, and settings the application can read.

The name and location of the application file depend on the application's host.
EXE-Hosted Applications
A configuration file of an application hosted by the EXE is in the same directory as the application. The name of the configuration file is the name of the application with a .config extension. For example, an application called myApp.exe can be associated with a configuration file called myApp.exe.config.

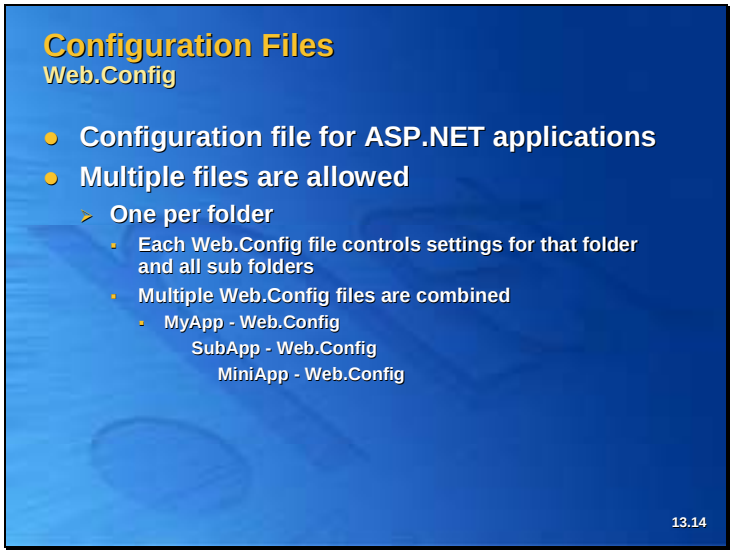
Internet Explorer-Hosted Applications

If an application hosted in Internet Explorer has a configuration file, the location of the configuration file is specified in a **<link>** tag with the following syntax:

```
<link rel="Configuration" HREF="[location]">
```

In this tag, [*location*] is a URL to the configuration file.

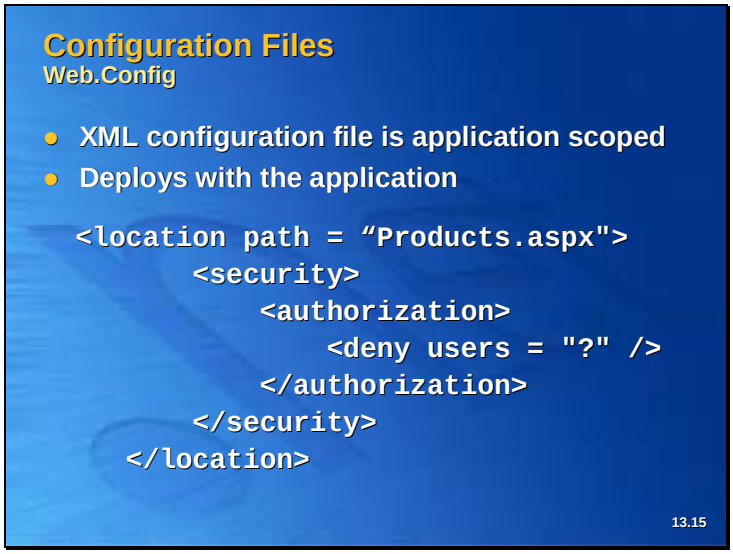
 ©2002 Microsoft Corporation



Configuration Files

Web.Config

ASP.NET Hosted Applications
ASP.NET configuration files are called **Web.Config** and they inherit the settings of configuration files in the URL path. For example, given the URL www.microsoft.com/aaa/bbb where www.microsoft.com/aaa is the Web Application, the configuration file that is associated with the application is located at www.microsoft.com/aaa. ASP.NET pages that are in the subdirectory /bbb will use both the settings that are in the configuration file at the application level and the settings in the configuration file that is in /bbb.

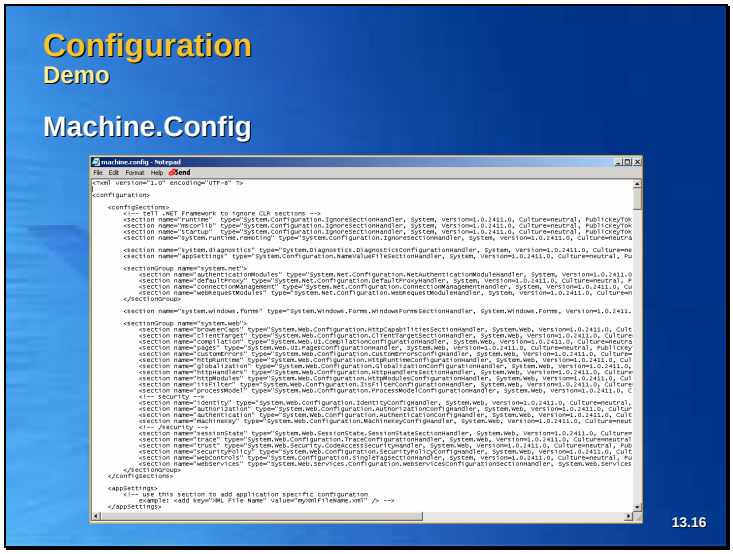


Configuration Files

Web.Config

Multiple configuration files, all named Web.Config, can appear in multiple directories on an ASP.NET Web Application server. Each Web.Config file applies configuration settings to its own directory and all child directories below. Configuration files in child directories can supply configuration information in addition to that inherited from parent directories, and the child directory configuration settings can override or modify settings defined in parent directories. The root configuration file named C:\WINNT\Microsoft.NET\Framework\ *version*\CONFIG\Machine.Config, provides ASP.NET configuration settings for the entire Web server.

 ©2002 Microsoft Corporation



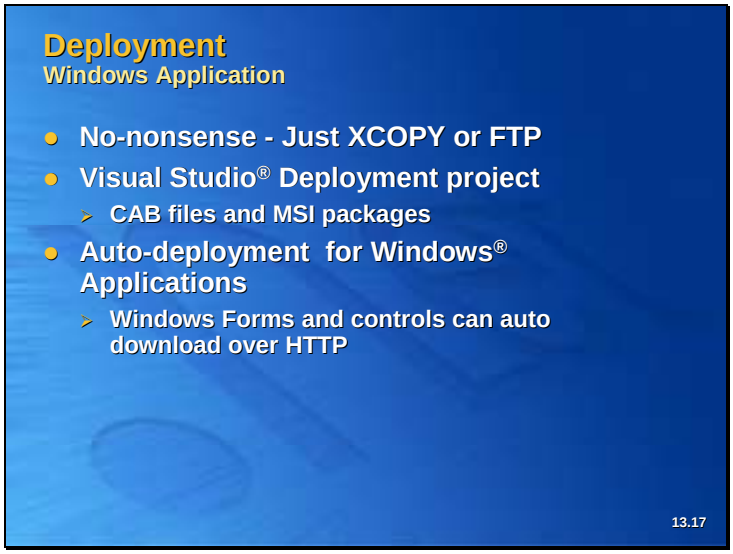
Configuration

Demo

Demonstrate how Machine.Config controls global server settings.

The purpose of this demo is to:

- Provide an overview of the Machine.Config file.
- Demonstrate various settings and configuration


Deployment
Windows Application
No-nonsense - Just XCOPY or FTP
Visual Studio® Deployment project
CAB files and MSI packages
Auto-deployment for Windows® Applications
Windows Forms and controls can auto download over HTTP
13.17

Deployment

Windows Application

The .NET Framework provides the following options for deploying applications:

- **Use XCOPY or FTP**

  Because CLR applications are self-describing and require no registry entries, you can use XCOPY or FTP to copy the application to an appropriate directory. The application then can be run from that directory.

- **Use an installer program**

  Use installer programs such as Windows Installer 2.0, to install, repair or remove Microsoft .NET Framework assemblies in the global assembly cache and in private directories.

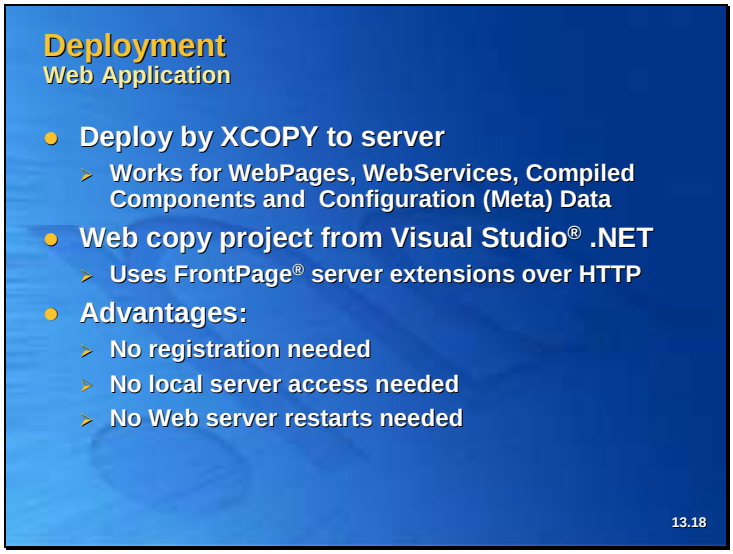
- **Use code download**

  Use code download when you are distributing your application over the Internet or through a corporate Intranet. You can download the code to a computer and then run the application.

- **Auto-deployment for Windows**

  Windows Applications can be downloaded by an HTTP server on an as-needed basis. This topic is discussed in Module 08: Creating Windows Applications.

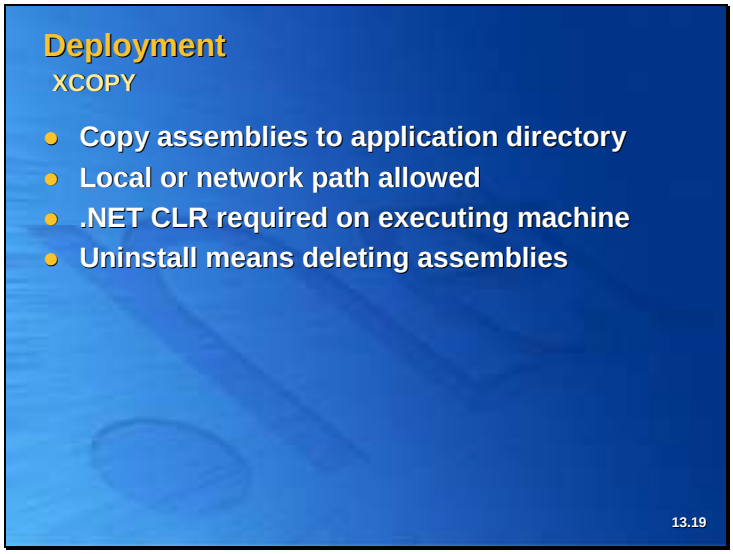
 ©2002 Microsoft Corporation


Deployment
Web Application
Deploy by XCOPY to server
Works for WebPages, WebServices, Compiled Components and Configuration (Meta) Data
Web copy project from Visual Studio® .NET
Uses FrontPage® server extensions over HTTP
Advantages:
No registration needed
No local server access needed
No Web server restarts needed
13.18

Deployment

Web Application

You can use **XCOPY** to deploy an ASP.NET application onto a server. You can then run one version of the application side-by-side with another version, and you can update the application without closing it. Because you control whether the application's components are shared, you can run the application simultaneously with other applications without DLL conflicts.

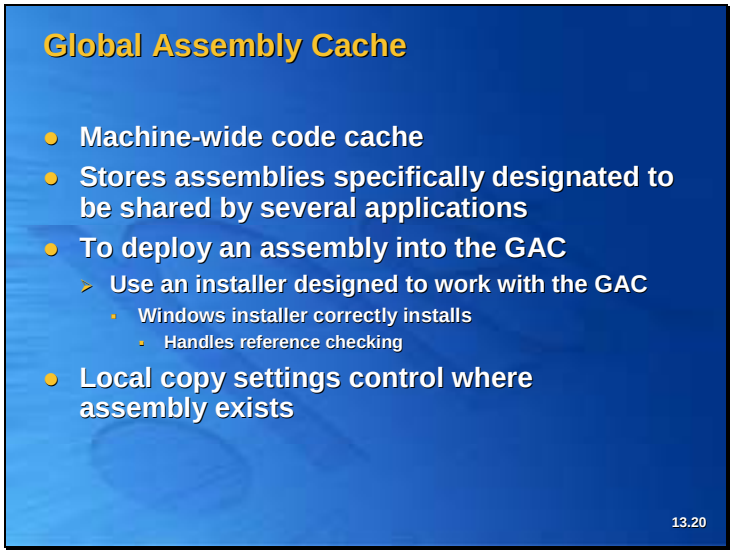


Deployment

XCOPY

XCOPY deployment and replication: Self-described and self-contained components and applications can be deployed without registry entries or dependencies.

 ©2002 Microsoft Corporation


Global Assembly Cache
Machine-wide code cache
Stores assemblies specifically designated to be shared by several applications
To deploy an assembly into the GAC
Use an installer designed to work with the GAC
Windows installer correctly installs
Handles reference checking
Local copy settings control where assembly exists
13.20

Global Assembly Cache

GAC

Each computer where the CLR is installed has a machine-wide code cache called the Global Assembly Cache (GAC). The GAC stores assemblies specifically designated to be shared by several applications on the computer.

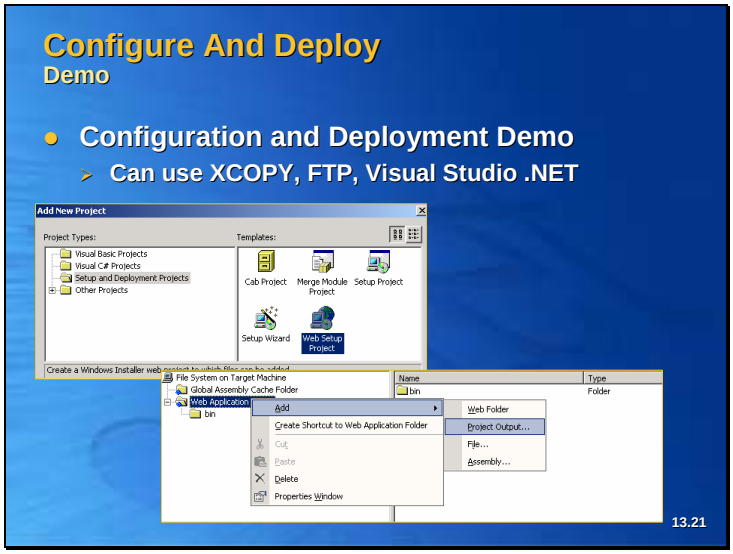
The GAC offers several advantages:

1. You can have one copy of an assembly that is shared by many applications.
2. Performance is better for assemblies in the GAC. When you install an assembly into the GAC, the signature is verified only at install time, while those not in GAC have the signature verified each time they run. Assemblies in the GAC also are located faster because the assembly name is also the folder name in the GAC. In addition, assemblies in the GAC can be optimized by the operating system.
3. The GAC is simply a folder structure that is easy to understand.

You must have Administrator privileges on a computer to install assemblies into the GAC.

There are scenarios where you do not want to install an assembly into the GAC. If you place one of your assemblies that make up an application in the GAC, you no longer can replicate or install the application by XCOPYing the application directory. You must move the assembly in the GAC as well.

In deployment scenarios, you should use Windows Installer 2.0 to install assemblies into the GAC. Use Windows Explorer or the Global Assembly Cache tool only in development scenarios because they do not provide assembly reference counting and other features provided when using the Windows Installer.



Configure and Deploy

Demo

Demonstrate how XCOPY can be used to roll out applications

The purpose of this demo is to:

- Demonstrate application folder tree
- Show how assemblies are stored and could be changed
- Discuss various XCOPY / Windows Explorer deployment

 ©2002 Microsoft Corporation

Deployment Using Installation Services

If you do not choose to install .NET Framework, you can set a condition flag to only install if the framework is installed.

Merge modules can be used to install common assemblies and files such as middle-tier components. Once you create a merge module, you can pull it into each separate setup that uses those business components, which allows you to assure that all setups contain the same files.

Custom Installers Deployment
Custom installers
Set up resources during deployment (e.g., create a message queue)
Run user code at install time (at the end of setup)
Dynamic properties
Useful for Web services
Property values can be changed after the project is built and deployed
RAD designer support to "bind" to configuration settings
13.23

Custom Installers Deployment

Server applications have pieces that lie outside of the EXE/DLL.

Custom installers ease the creation of these external dependencies at setup.

Dynamic properties simplify changing the pointers to these external dependencies as the application moves from development to testing to production.

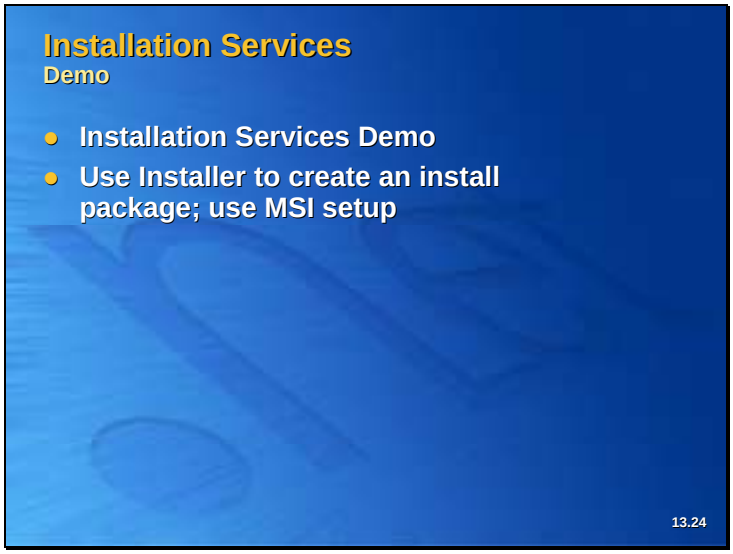
To dynamically set the properties of a component, set the dynamic properties for the component in the property window. Defining the dynamic properties creates an app.config file in the project that looks similar to the following:

```
<?xml version="1.0" encoding="Windows-1252" ?>
<configuration>
      <appSettings>
            <!--   User application and configured property settings go
here.-->
            <!--   Example: <add key="settingName" value="settingValue"/> -->
            <add key="SqlConnection1.ConnectionString" value="" />
      </appSettings>
</configuration>
```

The code within the application would read the appropriate key values from the app.config file before executing. The following demonstrates the code used to read the values from app.config file.

```
Me.SqlConnection1.ConnectionString =
CType(configurationAppSettings.GetValue("SqlConnection1.ConnectionString",
GetType(System.String)), String)
```

 ©2002 Microsoft Corporation



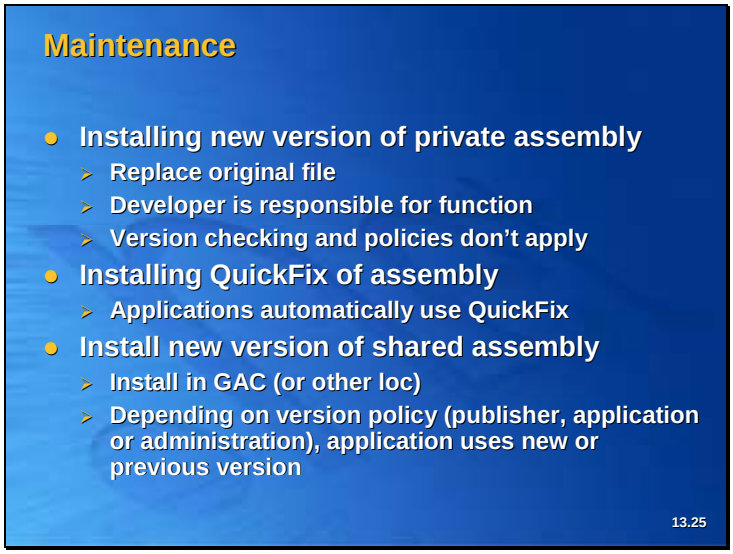
Installation Services

Demo

Demonstrate how to use the Windows Installer features in VS .NET to create and install a package.

The purpose of this demo is to:

- Show how to create a deployment package for a Web application using VS .NET
- Demonstrate how to construct and use the package


Maintenance
Installing new version of private assembly
Replace original file
Developer is responsible for function
Version checking and policies don't apply
Installing QuickFix of assembly
Applications automatically use QuickFix
Install new version of shared assembly
Install in GAC (or other loc)
Depending on version policy (publisher, application or administration), application uses new or previous version
13.25

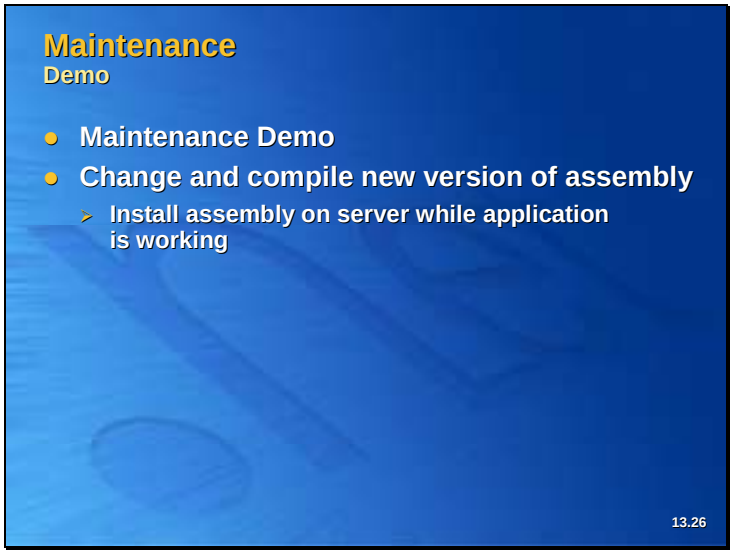
Maintenance

A QuickFix of an assembly doesn't automatically overwrite the existing copy of the assembly. You can ship a quick fix as a side-by-side version with a policy says that the applications should use it. This way, if the QuickFix breaks the application, the application has the opportunity to revert to the original version. It is recommended that different versions of the assembly always have different version numbers.

Quick Fixes don't have to go into the GAC; they can go into the application directory as well.

Install a new version of a shared assembly – By default, (in the absence of policy) the new application does not get picked up. This is a key factor in the process of making applications "not impactul" by default.

 ©2002 Microsoft Corporation



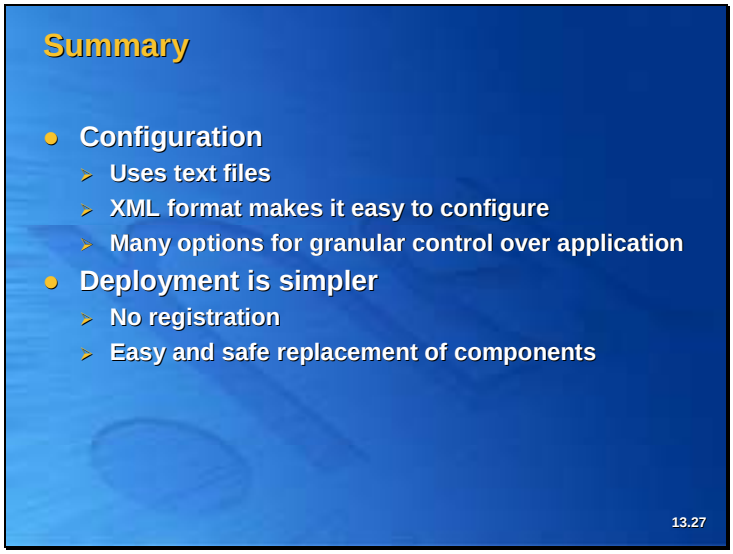
Maintenance

Demo

Demonstrate how to make changes and redeploy a new version of an assembly

The purpose of this demo is to:

- Show how to roll out new versions while application is running
- Demonstrate how application gracefully upgrades
- Demonstrate how this makes upgrading much easier



Summary

 ©2002 Microsoft Corporation

Resources
Basics of Distributed Applications - Multiple links about the pieces that make up a distributed application
http://www.gotdotnet.com/team/clr/about_distributedapps.aspx
13.28

Resources

Basics of Distributed Applications - Multiple links about the pieces that make up a distributed application: http://www.gotdotnet.com/team/clr/about_distributedapps.aspx

 ©2002 Microsoft Corporation